Model-Checking Frameworks: Outline

- Theory (Part 1)
  - Notion of Abstraction
  - Aside: over- and under-approximation, simulation, bisimulation
  - Counter-example-based abstraction refinement
- Abstraction and abstraction refinement in program analysis (Part 2)
  - Kinds of abstraction:
    - Data, predicate
  - Building abstractions
    - Aside: weakest precondition
  - Counter-example-based abstraction refinement

1

Outline, cont'd

- 3-valued abstraction and abstraction-refinement (Part 3)
  - 3-valued logic
  - Theory of 3-valued abstractions: combining over- and under-approximation
  - 3-valued model-checking
  - Building 3-valued abstractions
  - Counter-example-based abstraction refinement

2

Acknowledgements

The following materials have been used in the preparation of this lecture:

- Edmund Clarke
  - "SAT-based abstraction/refinement in model-checking", a course lecture at CMU
- Corina Pasareanu
  - Conference presentations at TACAS'01 and ICSE'01
- John Hatcliff
  - Course materials from Specification and Verification in Reactive Systems

Many thanks for providing this material!

3

Model Checking

- Given a:
  - Finite transition system $M(S, s_0, R, L)$
  - A temporal property φ
- The model checking problem:
  - Does M satisfy φ?

$$M \overset{?}{\models} \varphi$$

4

Model Checking

- Temporal properties:
  - "Always x=y"
    (**G**(x=y))
  - "Every Send is followed immediately by Ack"
    (**G**(Send → **X** Ack))

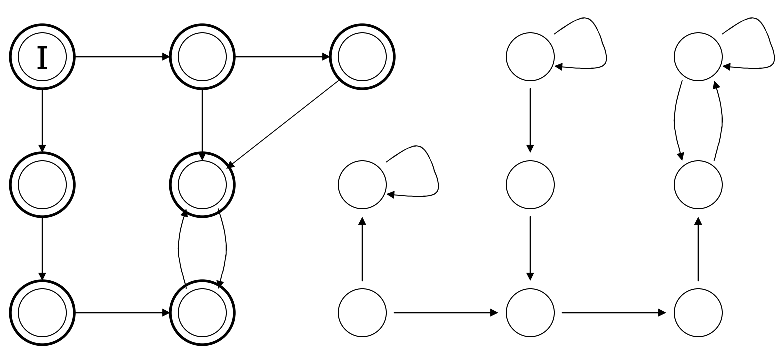
  "Safety" properties

  - "Reset can always be reached"
    (**GF** Reset)
  - "From some point on, always switch_on"
    (**FG** switch_on)

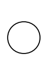
  "Liveness" properties

5

Model Checking (safety)

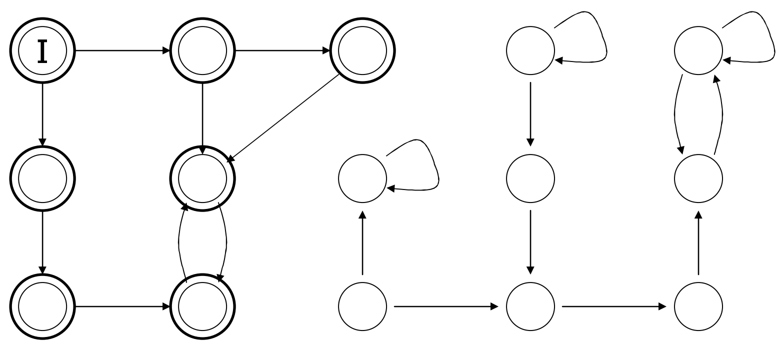


Add reachable states until reaching a fixed-point

 = bad state

6

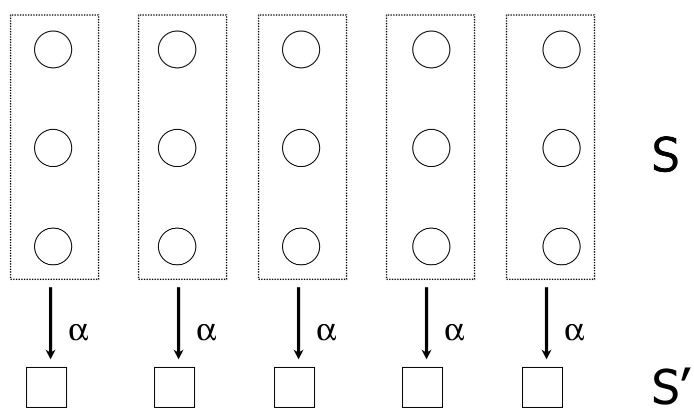
Model Checking (safety)



Too many states to handle !

= bad state

Abstraction



Abstraction Function $\alpha : S \rightarrow S'$

Abstraction Function: A Simple Example

- Partition variables into visible(□) and invisible(□) variables.
- The abstract model consists of □ variables. □ variables are made inputs.
- The abstraction function maps each state to its projection over □.

9

Abstraction Function: Example

$x1$	$x2$	$x3$	$x4$
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1

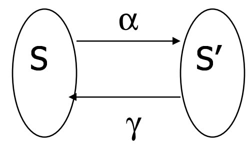
$\xrightarrow{\alpha}$

$x1$	$x2$
0	0

Group concrete states with identical visible part to a single abstract state.

10

Computing Abstractions



- S – concrete state space
- S′ – abstract state space
- α: S $\rightarrow$ S′ - abstraction function
- γ: S′ $\rightarrow$ S - concretization function
- Properties of α and γ:
  - $\alpha(\gamma(A)) = A$, for A in S′
  - $\gamma(\alpha(S)) \supseteq S$, for S in S
- The above properties mean that α and γ are Galois-connected

Aside: simulations

$M = (s_0, S, R, L)$

$M' = (t_0, S', R', L')$

Definition: p is a simulation between M and M' if

1. $(s_0, t_0) \in p$
2. $\forall\, (t, t_1) \in R'\ \exists (s, s_1) \in R$ s.t. $(s, t) \in p$ and $(s_1, t_1) \in p$

Intuitively, every transition in M' corresponds to some transition in M

Aside: bisimulation

M = (s_0, S, R, L)

M′ = (t_0, S′, R′, L′)

Definition: *p* is a bisimulation between *M* and *M′* if

1. *p* is a simulation between *M* and *M′* and
2. *p* is a simulation between *M′* and *M*

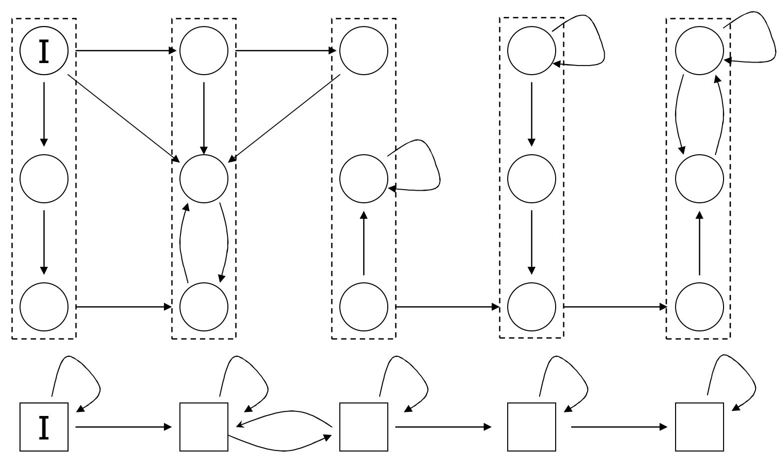
13

Computing Existential Transition Relation

- $R^{\exists\exists}$ [Dams'97]: $(t, t_1) \in R'$ iff $\exists\ s \in \gamma(t)$ s.t. $\exists\ s_1 \in \gamma(t_1)$ and $(s, s_1) \in R$
- This ensures that *M′* is the over-approximation if *M*, or *M′* simulates *M*.

14

Existential Abstraction (Over-Approximation)



15

Model Checking Abstract Model

- Let φ be a universally-quantified property (i.e., expressed in LTL or ACTL) and M' simulates M

- Preservation Theorem

$$M' \models \varphi \rightarrow M \models \varphi$$

- Converse does not hold

$$M' \not\models \varphi \not\rightarrow M \not\models \varphi$$

- The counterexample may be spurious

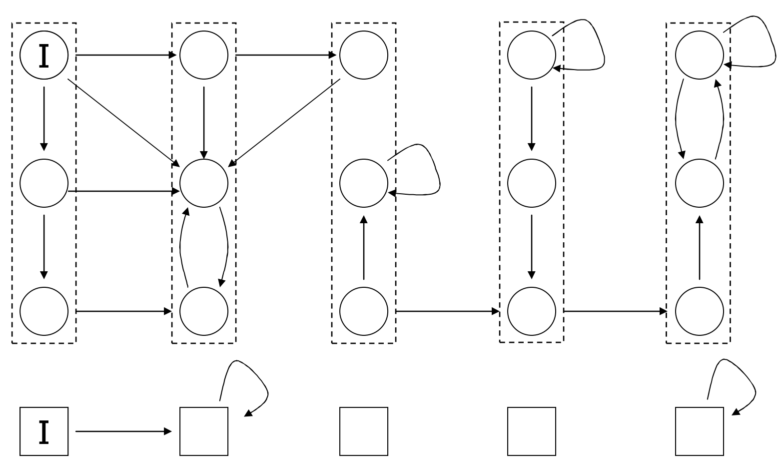
16

Computing Transition Relation

- $R^{\forall\exists}$ [Dams'97]: $(t, t_1) \in R'$ iff $\forall\, s \in \gamma(t)$ $\exists\, s_1 \in \gamma(t')$ and $(s, s_1) \in R$
- This ensures that *M'* is the under-approximation if *M*, or *M* simulates *M'*.

17

Universal Abstraction (Under-Approximation)



18

Model Checking Abstract Model

- Let φ be a existential-quantified property (i.e., expressed in ECTL) and M simulates M'
- Preservation Theorem

$$M' \models \varphi \rightarrow M \models \varphi$$

- Converse does not hold

$$M' \not\models \varphi \not\rightarrow M \not\models \varphi$$

19

Model Checking Abstract Model

$M = (s_0, S, R, L)$ and $M' = (t_0, S', R', L')$ related by bisimulation

Then, for every CTL/LTL property φ:

$$M' \models \varphi \rightarrow M \models \varphi$$
$$M' \not\models \varphi \rightarrow M \not\models \varphi$$

So, why not use bisimulation for abstraction?

20

Our specific problem

- Let φ be a universally-quantified property (i.e., expressed in LTL or ACTL) and M' simulates M

- Preservation Theorem

$$M' \models \varphi \rightarrow M \models \varphi$$

- Converse does not hold

$$M' \not\models \varphi \not\rightarrow M \not\models \varphi$$

- The counterexample may be spurious

21

Checking the Counterexample

- Counterexample : $(c_1, \ldots, c_m)$
  - Each c_i is an assignment to V.

- Simulate the counterexample on the concrete model.

22

Checking the Counterexample

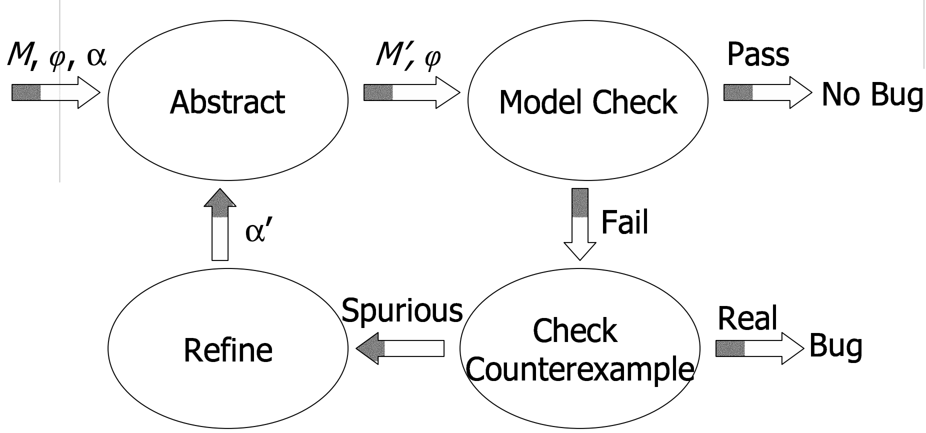
Concrete traces corresponding to the counterexample:

$\phi \;=\; I(s_1) \;\wedge$ (Initial State <- s_0 in our case)

$\bigwedge_{i=1}^{m-1} R(s_i, s_{i+1}) \;\wedge$ (Unrolled Transition Relation)

$\bigwedge_{i=1}^{m} \text{visible}(s_i) = c_i$ (Restriction of □ to Counterexample)

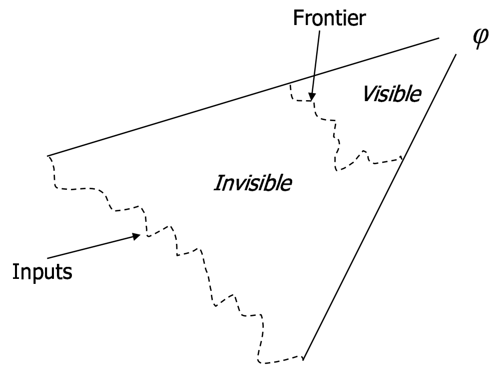
Abstraction-Refinement Loop



Refinement methods…

Localization

(R. Kurshan, 80's)



25

Refinement methods…

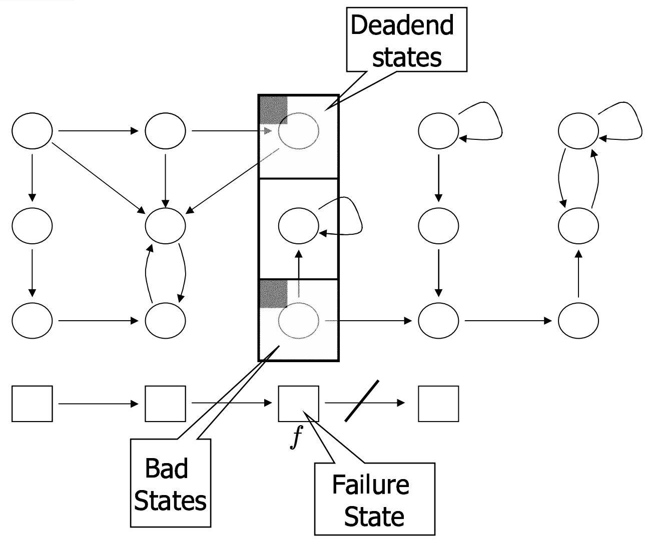
Abstraction/refinement with conflict analysis

(Chauhan, Clarke, Kukula, Sapra, Veith, Wang, FMCAD 2002)

- Simulate counterexample on concrete model with SAT
- If the instance is unsatisfiable, analyze conflict
- Make visible one of the variables in the clauses that lead to the conflict

26

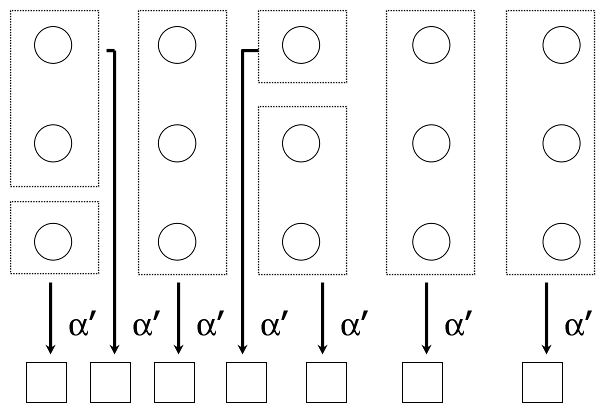
Why spurious counterexample?



Refinement

- Problem: Deadend and Bad States are in the same abstract state.
- Solution: Refine abstraction function.
- The sets of Deadend and Bad states should be separated into different abstract states.

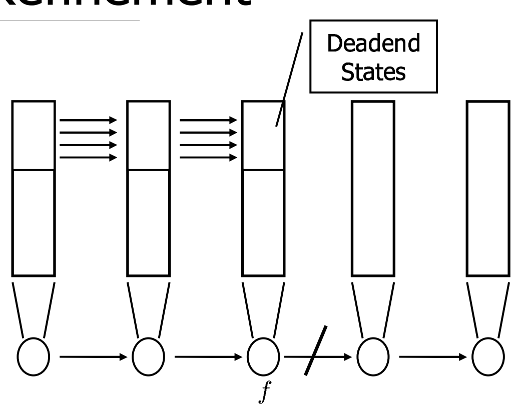
Refinement



Refinement : α′

29

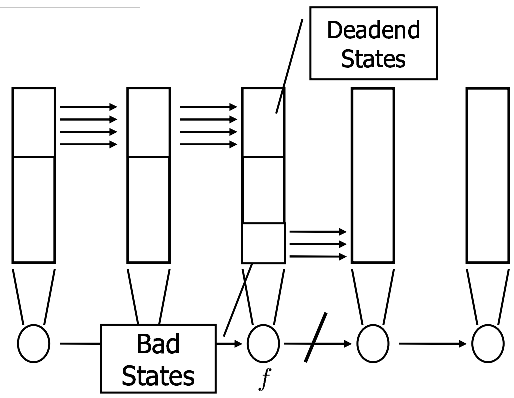
Refinement



$$\phi_D = I(s_1) \wedge \bigwedge_{i=1}^{f-1} R(s_i, s_{i+1}) \wedge \bigwedge_{i=1}^{f} \text{visible}(s_i) = c_i$$

30

Refinement



$$\phi_B = R(s_f, s_{f+1}) \wedge$$
$$\text{visible}(s_f) = c_f \wedge \text{visible}(s_{f+1}) = c_{f+1}$$

31

Refinement as Separation

d_1	0	1	0	0	1	0	1
b_1	0	1	0	0	0	1	0
b_2	0	1	0	0	1	1	1

□

□

Refinement : Find subset □ of □ that separates between all pairs of deadend and bad states. Make them visible.

Keep □ small !

32

Refinement as Separation

d_1	0	1	0	0	1	0	1
b_1	0	1	0	0	0	1	0
b_2	0	1	0	0	1	1	1

□

□

Refinement : Find subset □ of □ that separates between all pairs of deadend and bad states. Make them visible.

Keep □ small !

33

Refinement as Separation

<u>The state separation problem</u>

Input: Sets D, B

Output: Minimal □ ∈ □ s.t.:

$$\forall\, d \in D,\ \forall\, b \in B,\ \exists u \in \square.\quad d(u) \neq b(u)$$

The refinement α' is obtained by adding □ to □.

34

Two separation methods

- ILP-based separation
  - Minimal separating set.
  - Computationally expensive.
- Decision Tree Learning based separation.
  - Not optimal.
  - Polynomial.

We will not talk about these in class

35